Universidad Zaragoza
1542

Trabajo Fin de Grado

ANÁLISIS, DISEÑO Y DESARROLLO DE DASHBOARD PARA EL SISTEMA SMARTROADS DE ITERNOVA

ANALYSIS, DESIGN AND DEVELOPMENT OF DASHBOARD FOR ITERNOVA'S SMARTROADS SYSTEM

Autor
Marcos Canales Mayo

Director
Jorge Casas Cañada

Ponente
Raquel Trillo Lado

Escuela de Ingeniería y Arquitectura

2016

Análisis, diseño y desarrollo de Dashboard para el sistema SmartRoads de Iternova

RESUMEN

En este trabajo de fin de grado (TFG) se desarrolla un módulo que sirve de apoyo a la gestión de las carreteras, a petición del personal del proyecto *SmartRoads* desarrollado conjuntamente con la empresa Grupo Hermes Infraestructuras en México.

En este contexto, el TFG tiene como principal objetivo desarrollar un panel de control (*dashboard*) desde el que se puedan visualizar en tiempo real datos de interés, en su mayoría estadísticos, de módulos implementados en el sistema *SmartRoads*. En concreto, se requiere:

- Análisis, diseño y desarrollo de un nuevo módulo de *dashboard* para gestionar los *widgets* de los múltiples módulos a monitorizar.
- Una nueva funcionalidad para el módulo de *Google Charts* del núcleo del sistema (*core*), que permita la actualización de gráficas de forma asíncrona.
- Análisis e introducción de una nueva librería para hacer *scroll* horizontal.
- Análisis e introducción de una nueva librería para crear una malla (*grid*) de cajas/elementos que puedan manejarse con *drag and drop*.
- Implementación de los *widgets* correspondientes a los siguientes módulos:
  - Agenda de Tareas. Se requiere que se muestren las tareas del último año, desglosadas en tareas sin comenzar, empezadas y terminadas. Además, también se debe proporcionar un mapa donde se localicen todas estas tareas.
  - Estaciones Meteorológicas. Se requiere que se puedan visualizar los datos en tiempo real de las estaciones que el usuario final desee.
  - Estadísticas de Logs. Se requiere que se muestren datos sobre el uso del sistema por parte de los usuarios, considerando la información de los ficheros de *log*, como por ejemplo el número de visitas que reciben los 10 módulos más usados junto con sus porcentajes de uso.
  - Uso del disco del sistema. Se necesita mostrar información relativa al disco duro, como la cantidad de espacio que queda libre.

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Marcos Canales Mayo,

con nº de DNI 73161591-V en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Análisis, diseño y desarrollo de Dashboard para el sistema SmartRoads de Iternova,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 20 de Junio de 2016

Fdo: Marcos Canales Mayo

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

GLOSARIO DE SIGLAS

ACEX : Asociación de Empresas de Conservación y Explotación de Infraestructuras
AJAX : Asynchronous JavaScript And XML
API : Application Programming Interface
BSON : Binary JSON
DOM : Document Object Model
GPS : Global Positioning System
HTML : HyperText Markup Language
IDE : Integrated Development Environment
ITS : Intelligent Transport System
JSON : JavaScript Object Notation
LAMP : Linux, Apache, MySQL/MongoDB, PHP
MVC : Modelo-Vista-Controlador
PHP : PHP Hypertext Processor
PR : Publish Rate
RF : Requisito Funcional
RNF : Requisito No Funcional
SaaS : Software as a Service
SCADA : Supervisory Control And Data Acquisition
SGBD: Sistema Gestor de Base de Datos
TTR : Time To Refresh

1. Introducción

1.1 Entorno laboral

ITERNOVA S.L., es una empresa con sedes en Zaragoza, Teruel y Mexico DF que se encarga de proveer soluciones tecnológicas innovadoras para empresas, ciudadanos y administradores. Actualmente su principal campo de trabajo es el desarrollo del sistema *SmartRoads*. Se trata del primer sistema web *SCADA* (*Supervisory Control And Data Acquisition*) desarrollado para la gestión de la Conservación y Explotación de Carreteras. En este campo se enmarca el presente TFG, que se ha desarrollado íntegramente en los servidores de pruebas de la empresa, incluyendo el depurado de errores.

ITERNOVA S.L apuesta por la continua innovación, lo que les ha permitido desarrollar productos de éxito e implantación a nivel nacional, como el sistema *SmartRoads.* Gracias a ello la empresa recibió el VIII Premio nacional ACEX 2012 a la Seguridad en Conservación de carreteras, además de otros galardones como el Premio Sociedad de la Información Aragón 2012 Empresa Junior del año y Premio Empresa Teruel Innovación 2012.

1.2 Antecedentes y Estado del Arte

El sistema *SmartRoads* es el primer Sistema Web *SCADA* desarrollado para la gestión de la Conservación y Explotación de Carreteras, e implantado actualmente a nivel nacional.

El sistema plantea un importante salto cualitativo en la gestión de la explotación de carreteras, permitiendo acceder a toda la información de las mismas de una forma integrada gracias, principalmente, a las siguientes características:

- Se trata de un Sistema Web **multidispositivo en código libre** (no requiere de licencias), modular (se adapta a las necesidades de cualquier demarcación) y multiusuario (con diferentes roles de acceso a la información).
- El sistema está basado en el concepto de **SaaS** (*Software as a Service*), pudiendo ser adaptado a las necesidades de cada cliente.

Además, el sistema integra de una forma optimizada la gestión de todos los elementos de interés de las carreteras y está formado por los siguientes módulos:

- Módulo de **información y gestión general** de carreteras, para gestionar los elementos más significativos inventariados y geolocalizados. En este módulo se pueden consultar las instalaciones y elementos de la carretera, fotografías aéreas en alta resolución y cartografía de *Google*

Maps. También es posible identificar todos los puntos con restricciones de galibo, anchura, estructuras. Además, se pueden listar los puntos de interés para gestionar la vialidad en caso de cortes u otros problemas en la carretera (cruces de mediana, intersecciones, desvíos...).

- Módulo de **gestión de la vialidad** en tiempo real, para gestionar flotas mediante *GPS* (*Global Positioning System*), cámaras de explotación, aforos, pesaje de vehículos, paneles de información, etc. Esto es gracias a que el sistema incorpora sistemas *ITS* (*Intelligent Transport Systems*) en tiempo real.

- Módulos de **comunicación** (entre gestor de la carretera y ciudadanos) y de **documentación**, que integran un gestor de expedientes y funcionalidades avanzadas de gestión.

- Módulo de **agenda de tareas**, que incluye un sistema de control y seguimiento de trabajos para la gestión de las actuaciones tanto programables como no programables en vialidad. Este módulo permite registrar todas las tareas de vialidad que se llevan a cabo en la carretera y las muestra en función de prioridades establecidas por la carta de servicios al ciudadano. Cada servicio es acompañado de sus protocolos de actuación y de unos mínimos de calidad establecidos (como por ejemplo un umbral de tiempo de respuesta ante las incidencias).

- Módulo de **estaciones meteorológicas** en el que se pueden observar los datos de temperatura, viento, etc tomados por las estaciones en distintos puntos de las carreteras. Esto es útil, por ejemplo, para determinar si la temperatura en una carretera ha superado el punto de rocío y así poder actuar en consecuencia.

- Módulo de **estadísticas de *logs***, para recopilar datos estadísticos y generar informes de todas las acciones que los usuarios llevan a cabo en el sistema *SmartRoads*.

- Módulo de **gestión económica**, para seguir programas de trabajo (contratos), gestión financiera (dinero invertido) y gestión física (avance de las obras).

- Módulo para **gestionar las actividades de seguridad vial**, en el que se gestionan los accidentes con víctimas que se producen en la vía. A través de este módulo se generan informes de accidentalidad de los distintos tramos de carretera (que indican por ejemplo la frecuencia de incidencias de un tipo concreto y el tiempo de respuesta a esas incidencias) para poder tomar las medidas oportunas que consigan reducirlos.

- Módulo de **archivos**, para mostrar datos de uso de disco del sistema, como por ejemplo la cantidad de espacio libre que queda.

- **Aplicación móvil**, que facilita la toma de datos por parte de los operarios de campo y su introducción en los módulos del sistema.

Cualquier usuario, haciendo uso de un dispositivo con conexión a la red, puede consultar toda la información de la carretera accediendo con su nombre de usuario y contraseña o con su certificado digital.

Gracias a la modularidad del sistema de gestión *SmartRoads*, las diferentes secciones (videocámaras, agenda de vialidad, gestión de flotas *GPS*, estaciones meteorológicas, paneles informativos, fotografías aéreas...) se integran en función de las **necesidades concretas del cliente**. Se puede comenzar con las herramientas básicas y, posteriormente, ir ampliando según aumenten las necesidades. Por ello, es una plataforma que se adapta y personaliza tanto a pequeños sectores como a grandes demarcaciones de carreteras.

El sistema *SmartRoads* es, en conclusión, un sistema dedicado a la **gestión de la Explotación y Conservación de carreteras**, que está siendo implantado paulatinamente en las principales provincias y comunidades autónomas españolas, tales como Aragón, Murcia, Soria, Valencia, Tarragona y Castilla La Mancha entre otras, así como en las principales carreteras y autopistas de México y en empresas concesionarias dedicadas a la conservación y la explotación de carreteras.

1.3 Objetivos

El objetivo principal de este TFG es el **análisis, diseño y desarrollo de un panel de control** (*dashboard*) para monitorizar en tiempo real múltiples módulos del sistema *SmartRoads* de la empresa Iternova S.L. El objetivo final radica en la **integración y puesta en producción** del módulo en el sistema *SmartRoads*. Adicionalmente, se requiere implementar el contenido de los *widgets* de los siguientes módulos del sistema: Archivos, Estadísticas de Logs, Agenda de Tareas y Estaciones meteorológicas.

Para cumplir estos objetivos es necesario actualizar otros módulos, librerías y utilidades del sistema. En concreto, es necesario que:

- La utilidad del sistema que genera gráficas usando la librería de *Google Charts* permita que se actualicen las gráficas por *AJAX* (*Asynchronous JavaScript And XML*).
- Se introduzca una nueva librería que permita hacer *scroll* horizontal para rotar automáticamente el contenido de los *widgets* (por ejemplo para mostrar una estación meteorológica cada 5 segundos o para alternar entre tabla, gráfica y mapa del *widget* de Agenda de Tareas).
- Se sustituya la librería que gestiona los *grids* de elementos por otra más moderna, debido a que la actual tiene limitadas funcionalidades.

En resumen, como resultado final, se pretende que, una vez terminado el TFG, el sistema *SmartRoads* disponga de un panel de control **completamente configurable** por el usuario para poder visualizar en tiempo real los datos más relevantes de otros módulos del sistema.

1.4 Estructura de la memoria

A partir de este punto, la memoria se estructura en los siguientes puntos:

- **Capítulo 2. Análisis de Requisitos:**
  Estudio de las necesidades que se van a satisfacer con este módulo.
- **Capítulo 3. Diseño:**
  Planteamiento de la solución a estas necesidades.
- **Capítulo 4. Implementación:**
  Cómo se ha llevado a cabo esta solución, desde el entorno de desarrollo hasta las metodologías y tecnologías empleadas.
- **Capítulo 5. Tests unitarios:**
  Pruebas que verifican, en la medida de lo posible, la corrección de la solución adoptada.
- **Capítulo 6. Resultados:**
  Exposición de los resultados obtenidos tras la realización de este TFG.
- **Capítulo 7. Conclusiones:**
  Análisis y valoración de la utilidad del proyecto tanto para el sistema en el que se ha desarrollado como en el ámbito personal y profesional.
- **Capítulo 8. Bibliografía:**
  Fuentes de utilidad durante el transcurso del proyecto.
- **Capítulo 9. Anexos:**
  Detalles que pueden ser de interés, entre los que se incluyen:
  - Pantallas de configuración del panel de control
  - Pantallas de visualización del panel de control
  - Manual de ayuda para el usuario: instrucciones de uso del *dashboard*.
  - Documentación para el desarrollador: para que, en ocasión de que se necesite, se disponga de documentación relacionada con la implementación del módulo.
  - Recopilación de esfuerzos durante el desarrollo del proyecto.

2. Análisis de Requisitos

En este apartado se describen los requisitos que debe satisfacer el panel de control. Se diferencian entre requisitos funcionales (RF), es decir, funcionalidades del *dashboard*, y requisitos no funcionales (RNF), como restricciones de implementación. Además, se describen las principales herramientas del sistema *SmartRoads* que se van a emplear durante el desarrollo del *dashboard*.

2.1. Dashboard

Tal y como se ha comentado en la introducción, uno de los principales objetivos es mostrar la información de los módulos en tiempo real. Por lo tanto, uno de los principales requisitos es que el contenido de los diferentes *widgets* debe de **mantenerse actualizado** (código RF1). Decidir el cómo se actualiza la información implica adelantar en cierta medida la etapa de diseño, dado que existen dos técnicas que condicionan este requisito:

- *Pull*: se trata de la técnica más conocida para este tipo de situaciones. Se caracteriza por ser el cliente el que inicia la comunicación. Consiste en, con un tiempo de refresco determinado (*Time to Refresh -TTR-*), solicitar nueva información al servidor y actualizar. Esta metodología se conoce como *polling* (encuesta).
- *Push*: en este caso, la comunicación empieza desde el servidor, cuando hay un cambio de estado, es decir, una actualización de la información que hay que comunicar a todos los clientes que están visualizándola.

Ambas técnicas tienen sus ventajas e inconvenientes. Si se usa la técnica *pull*, en caso de tener un *TTR* reducido, la información se mantendrá actualizada con un margen muy pequeño de error, pero puede dar lugar a peticiones innecesarias al servidor si la frecuencia con la que se actualiza la información (*Publish Rate -PR-*) también es pequeña, y por consiguiente a un **alto tráfico en la red**. Por contra, si el *TTR* es grande, puede que el cliente **pierda actualizaciones** de la información, por lo que no se mantendría actualizado al 100%. De esto se deduce que el *TTR* ideal debe de ser igual a 1/*PR* (puesto que *TTR* es período y *PR* es frecuencia). La *figura 2.1* contiene ejemplos en los que se producen los problemas descritos anteriormente.

En cambio, al usar la técnica *push* se asegura la consistencia de los datos en el lado del cliente respecto al servidor, dado que el servidor envía la nueva información siempre que cambia de estado. Una de las desventajas de este método es que la conexión entre cliente y servidor siempre debe de estar abierta, ya que sino, si el servidor cambia de estado debería de esperar a que el cliente hiciera una nueva petición para comunicarle los cambios que ha habido en la información (cosa que haría que esta técnica fuera equivalente al *pull* en cuanto a resultados). Otra desventaja es que requiere una mayor

complejidad para ser implementada, además de tener que generar, almacenar y realizar seguimiento de estado de los clientes a los que enviar la información actualizada.

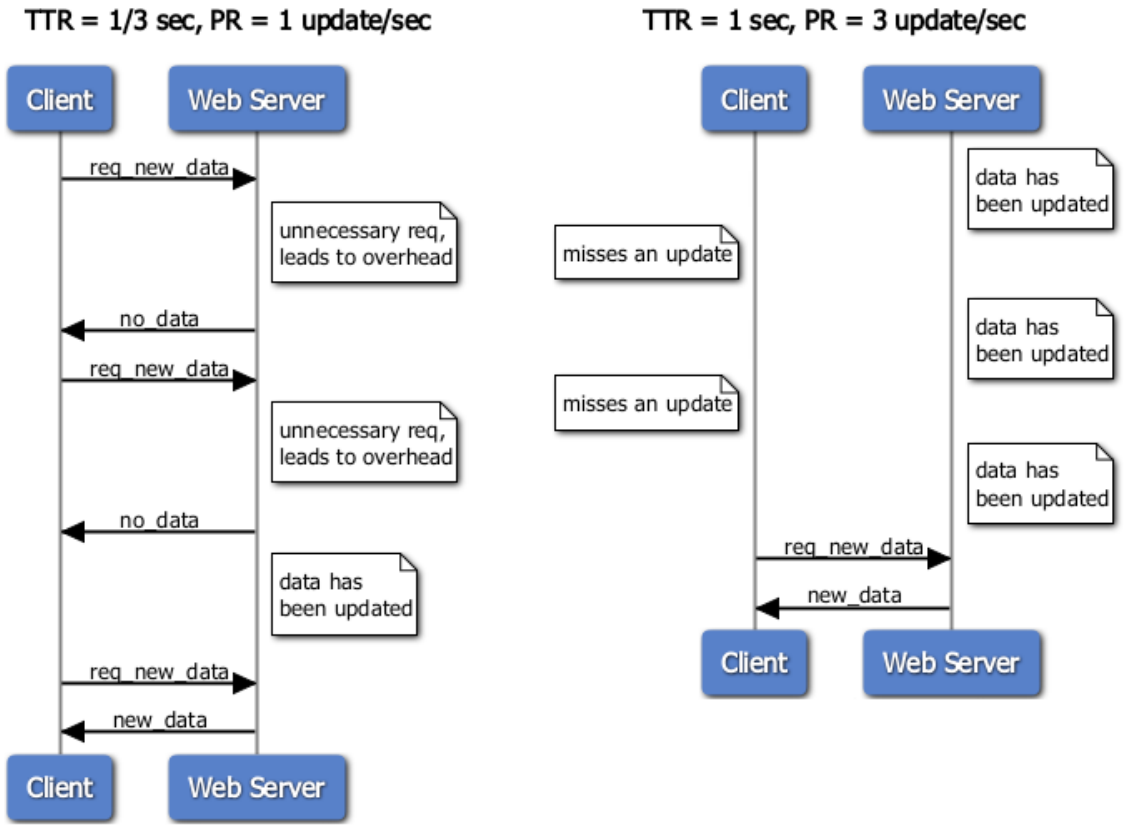


Figura 2.1. Problemática asociada a los conceptos TTR y PR

Una vez conocidas las dos técnicas, hay que analizar diversos factores:

- La frecuencia con la que los módulos a monitorizar actualizan su información puede ser fija o variable (dependiendo del módulo). Por ejemplo, el módulo de Estadísticas de Logs tiene un periodo de actualización fijo de un día, mientras que el módulo de Agenda de Tareas puede tener nueva información en cualquier instante.
- La frecuencia de actualización en el lado del cliente va a ser relativamente pequeña, ya que no es crítica para el correcto funcionamiento del *dashboard*. Esto significa que no es necesario que los *widgets* se actualicen en cuanto el servidor reciba nueva información.
- No se tiene experiencia con el uso de tecnologías *push* (cosa que llevaría a un número de horas extra de trabajo relevante para investigar en este campo).

Por todo ello, se llega a la conclusión de que es más conveniente y sencillo implementar y usar la **técnica *pull* para garantizar la consistencia de los datos** en el cliente respecto al servidor.

Otro de los requisitos principales es que cada usuario podrá configurar el *dashboard* a su medida, pudiendo escoger qué módulos (de los disponibles en el sistema) quiere monitorizar, y configurarlos a su gusto (RF2). Además, los módulos del *dashboard* deben de tener al menos unas determinadas opciones de configuración, **comunes** a todos los módulos. Estas opciones (RF3) son el tiempo de refresco de la información mostrada (dado que hemos escogido usar la técnica *pull*) y la *setup*, que indica las formas en las que se representará la información del módulo (en tablas, gráficas, mapa...). Opcionalmente, podrá haber **parámetros de configuración específicos** a cada módulo (RF4). Por ejemplo, las estaciones que se quieren visualizar en el módulo de Estaciones Meteorológicas o los sectores a filtrar en el módulo de Agenda de Tareas.

Respecto a la visualización del panel de control, es necesario que el usuario pueda ocultar/mostrar los *widgets* individuales (RF5). Además, también podrá alterar el orden en el que se muestran los *widgets*, es decir, su posición (RF6). El orden en el que inicialmente se muestran los *widgets* debe de ser el mismo que el usuario configuró la vez anterior (RF7).

Adicionalmente, se requiere la implementación de los siguientes *widgets*:

- Agenda de Tareas (RF8), donde se muestren en un mapa las tareas que cumplan las siguientes condiciones: tareas no iniciadas, iniciadas sin finalizar o ejecutadas (terminadas). Además, se debe incluir una gráfica y una tabla que resuman en términos mensuales y anuales la cantidad de tareas que ha habido de cada tipo.
- Estaciones Meteorológicas (RF9), donde se muestre información de las estaciones meteorológicas previamente seleccionadas por el usuario. En esta información se deben incluir datos de posición de la estación meteorológica y la evolución temporal de la temperatura medida respecto al punto de rocío, en forma de tabla y de gráfica.
- Estadísticas de Logs (RF10), donde se muestre el número de accesos a los distintos módulos del sistema, desglosado en meses, en forma de tabla y de gráfica.
- Uso de disco en el sistema (RF11), donde se muestre una tabla con información básica sobre el uso del sistema, como por ejemplo el número de archivos subidos, porcentaje de uso del disco, cantidad de espacio libre, espacio total en disco, etc.

2.2. Otros Requisitos

Otros requisitos adicionales son:

- Usabilidad (RNF1). El panel de control debe de tener una interfaz intuitiva para el usuario. De esta forma, el usuario debe de ser capaz de configurarlo a su medida de forma cómoda.
- Responsive (RNF2). El diseño debe de ser adaptable a cualquier dispositivo (móviles, tablets, pcs...).
- Escalabilidad y mantenibilidad (RNF3). Debe de ser fácil la introducción de nuevos módulos en el *dashboard* para el desarrollador.
- Reusable (RNF4). Debe permitirse la posibilidad de reusar código para otras funcionalidades.

Además, para algunos *widgets* del panel de control conviene disponer de un *scroll* horizontal que rote automáticamente, por ejemplo, las distintas representaciones de datos (tabla, gráfica, mapa, etc) que se han configurado para el módulo. También puede servir para rotar la información de las estaciones en el módulo de Estaciones Meteorológicas. Por esta razón es necesario introducir en el *core* una nueva librería que proporcione esta funcionalidad (RF12).

También es necesario actualizar el módulo del *core* que permite pintar gráficas, que usa la librería de *Google Charts*. Para el panel de control, es necesario que las gráficas se actualicen de forma asíncrona, pero el controlador de este módulo no dispone de esta característica, por lo que se requiere implementarla (RF13).

El último requisito afecta al *plugin Shapeshift* [11] de la librería *jQuery*, que se emplea en el sistema *SmartRoads*. Este *plugin* se emplea para gestionar mallas (*grids*) de elementos (en este caso *widgets*). Dado que el sistema tenía una versión obsoleta de este *plugin* y además proporcionaba limitadas funcionalidades, es necesario sustituirlo por otra herramienta que cumpla las necesidades de forma **simple y eficaz** (RF14).

2.3 Herramientas del Sistema

Siguiendo la filosofía de Iternova de usar software libre, el desarrollo de este proyecto se enmarca en un sistema *LAMP*, es decir, *GNU/Linux* como sistema operativo, *Apache* como servidor web, *MySQL* [1] y *MongoDB* [2] como bases de datos y *PHP5* (*PHP Hypertext Processor*) [3] como lenguaje de programación y motor de aplicación; en combinación con otros lenguajes de programación, como *JavaScript* (para crear dinamismo y gestionar comunicación con *AJAX*, sobre todo usando la librería *jQuery* [4]) y de *APIs* (*Application Programming Interface*) externas como *Google Maps* [5] o *Google Charts* [6].

En los siguientes puntos se detallan algunas de las herramientas y tecnologías más utilizadas en el *framework* desarrollado por Iternova.

2.3.1 MongoDB

Es un sistema gestor de base de datos (SGBD) no relacional multiplataforma que almacena los datos en formato *JSON* (*JavaScript Object Notation*) [7] binario, también denominado *BSON* (*Binary JSON*) [8]. Es el principal sistema de base de datos que usa el sistema *SmartRoads*, y en este TFG se usará para almacenar la configuración del *dashboard* de cada usuario.

2.3.2 MySQL

Es un sistema de base de datos relacional multiplataforma. En el sistema *SmartRoads* se usa en los módulos antiguos, como el de Agenda de Tareas, ya que en los comienzos del sistema no existía *MongoDB*. No obstante, es posible que en un futuro se plantee una migración progresiva a la base de datos no relacional.

MySQL no se utilizará específicamente en el *dashboard*, pero se requiere su uso en el módulo de Agenda de Tareas, puesto que este módulo contiene toda la información en este sistema gestor de base de datos. Por lo tanto, a lo largo de este TFG, MySQL se usará sólo para recuperar datos realizando consultas.

2.3.3. Gearman

Gearman [9] es una aplicación diseñada para gestionar colas de tareas, que equilibra la carga de trabajo en **segundo plano** entre varias máquinas en paralelo, con el objetivo de no sobrecargar el servidor. Está orientado a todo tipo de aplicaciones.

Ejemplos de su uso son, el proceso de subir una foto en *Instagram* o las notificaciones de eventos en tiempo real en *Facebook*. El uso de *Gearman* también puede orientarse a ejecutar determinadas tareas en un entorno

optimizado para ellas. Esto puede ocurrir cuando un servidor web desea ejecutar una tarea para la cual no está optimizado. En este caso puede asignar la ejecución de esta tarea a otro servidor con otro tipo de entorno que esté optimizado para ello, ya sea porque tiene otra arquitectura *hardware*, otro sistema operativo u otro lenguaje de programación más adecuado.

En el sistema *SmartRoads* se usa *Gearman*, por ejemplo, para importar grandes cantidades de datos, de manera que la ejecución de esta tarea en segundo plano permite al usuario continuar usando el sistema sin tener que bloquearse esperando a que termine. Además, se emplea para que las tareas se completen lo antes posible, debido a que con *Gearman* se puede **distribuir la carga de trabajo** entre varios servidores, consiguiendo ejecutar varias tareas en paralelo. *Gearman* también se emplea junto con *Cron* [10] (descrita a continuación), para generar y cachear en el sistema los datos de algunos *widgets* periódicamente (debido al elevado coste de su cálculo).

Para conocer más detalles sobre el funcionamiento de esta herramienta ver el anexo I.

2.3.4. Cron

Cron es una herramienta integrada de forma nativa en los sistemas *Linux* para **ejecutar tareas de forma periódica**. El sistema *SmartRoads* tiene integrado un controlador que usa esta herramienta para ejecutar las tareas periódicas de todos sus módulos. En ocasiones se combina el uso de *Gearman* con *Cron* para ejecutar tareas periódicas que supongan una gran carga de trabajo.

En este TFG, como se ha mencionado anteriormente, se usa *Cron* para generar los datos de algunos *widgets* de forma periódica empleando *Gearman*.

3. Diseño

En el presente capítulo se explican los aspectos más relevantes de la solución propuesta para satisfacer los requisitos previamente descritos.

Se ha optado por una solución en que el panel de control tenga dos pantallas, una para la configuración y otra para la visualización. En la pantalla de configuración se muestran los distintos módulos que se pueden visualizar, de manera que el usuario pueda escoger los que él desee y configurarlos a su medida. Por otra parte, en la pantalla de visualización del *dashboard* se muestran los distintos módulos que han sido seleccionados anteriormente en forma de *widgets*. El usuario puede ocultarlos/mostrarlos y ordenarlos (con *drag and drop*) como él prefiera.

3.1. Arquitectura Modelo-Vista-Controlador

El sistema al completo se basa en el patrón Modelo-Vista-Controlador (*Model-View-Controller*), por lo que es necesario comprender su funcionamiento y posteriormente plantear el esquema del panel de control. Este patrón se emplea para separar la capa de datos, su control o procesamiento y los aspectos visuales y de interfaz con el usuario.

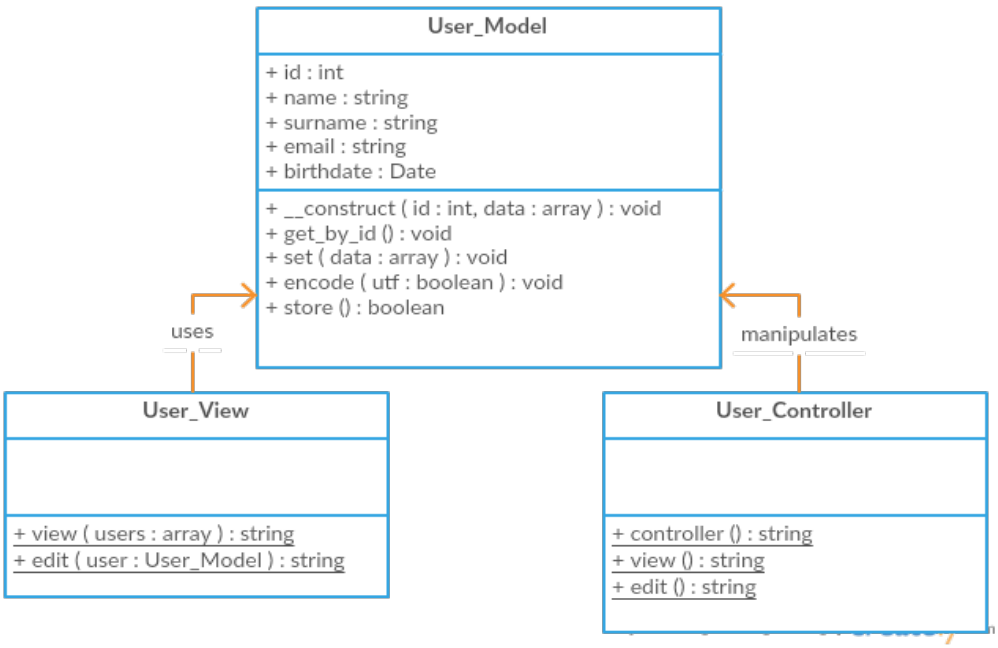


Figura 3.1. Ejemplo de patrón MVC

En la *figura 3.1* se muestra un diagrama de clases que muestra a grandes rasgos cómo se usa este patrón en el sistema *SmartRoads*. En concreto, este ejemplo muestra cómo se usa el patrón para desarrollar un

módulo que permite mostrar y editar usuarios. En el diagrama se aprecian tres clases con distintas funciones:

- *Model*. Esta clase representa el **modelo lógico**. Por ello, existen métodos para cargar el modelo desde base de datos (en este ejemplo es *get_by_id*) o a partir de datos pasados por parámetro en la petición del cliente (en el ejemplo es el método *set*). Siempre tiene los métodos *encode,* para codificar los datos en sus correspondientes formatos (*string*, *int*, *array*...), y *store,* para actualizar el objeto en la base de datos.
- *Controller*. Esta clase actúa como intermediaria entre el cliente y el servidor. Todas las solicitudes llegan al método *controller*, que, dependiendo de los parámetros de la petición hará una cosa u otra (en este ejemplo puede editar o visualizar usuarios). Las funciones de esta clase sólo tienen tareas de **procesamiento** de datos (incluyendo modificar el modelo lógico y almacenarlo en base de datos).
- *View*. Esta clase tiene como único objetivo generar el código *HTML* que se visualizará en la **interfaz** del usuario. Por lo tanto, necesita los datos del modelo lógico (p. ej. para mostrar la lista de usuarios necesita sus nombres, apellidos...).

3.2. Comunicación asíncrona cliente - servidor

Dado que el panel de control desarrollado es de tiempo real, el concepto de comunicación asíncrona adquiere relevancia. Esto es debido a que, una vez el usuario cargue la pantalla de visualización del *dashboard*, es necesario que se vayan actualizando periódicamente todos los *widgets*. La solución a esto es que cada *widget* gestione la actualización de sus datos, por tanto al menos habrá una petición *AJAX* por cada uno de los *widgets*.

La opción más sencilla sería que el servidor envíe todo el código *HTML* del contenido del *widget* y que el cliente lo cargue. Sin embargo, esta opción no proporciona una **buena experiencia de usuario** porque el sistema requiere la actualización de mapas y gráficas que tardan un tiempo notable en ser recargadas. Para reducir el peso de los datos enviados y agilizar la actualización de datos dinámicamente en el lado del cliente, se plantea el uso del formato *JSON* en la transferencia de datos desde el servidor al cliente. Además, gracias a su legibilidad, el código resulta fácil de mantener. En la *figura 3.2* se muestra un diagrama de secuencia que muestra el flujo de comunicación existente entre cliente y servidor para actualizar periódicamente un *widget*, una vez el cliente ya ha cargado la página de visualización del *dashboard*. Dado que cada *widget* puede tener un tiempo de refresco distinto, el diagrama de secuencia se debe de aplicar a cada uno, por lo que existirán al menos tantas *polls* como *widgets*.

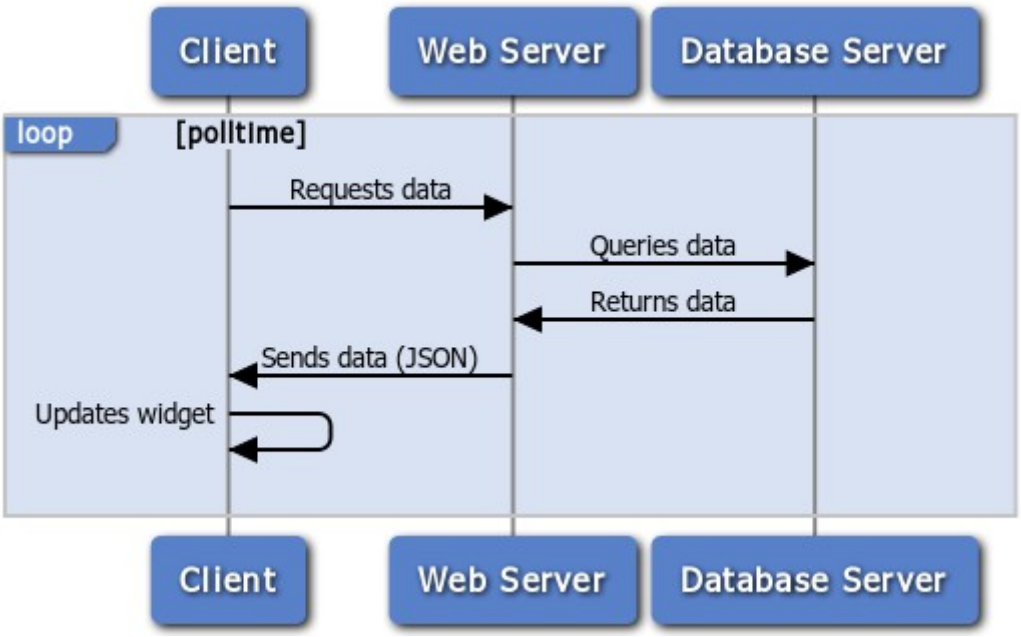


Figura 3.2. Actualización periódica de un widget

3.3 Handlers

Para favorecer el **mantenimiento, escalabilidad y configuración** del *dashboard* es necesario utilizar *handlers*. Los *handlers* (también llamados enrutadores, manejadores o controladores) permiten llamar a métodos de clases específicas en función de una acción genérica que recibe como parámetro y de unas opciones fijadas en un fichero de configuración. Gracias a la estructura de los *handlers* resulta muy sencillo añadir una nueva funcionalidad. Esto es debido a que sólo es necesario crear una nueva clase con las acciones específicas a realizar (que serán llamadas desde el controlador genérico) y modificar los parámetros de configuración.

El uso de *handlers* es necesario en los dos componentes (pantallas) del *dashboard* debido a las siguientes razones:

- En el componente de configuración, ya que cada *widget* tiene unos parámetros de configuración específicos que los demás no tienen.
- En el componente de visualización, pues cada *widget* muestra un contenido distinto al de los demás.

En la *figura 3.3* se muestra una imagen que muestra el concepto de los *handlers*. La *figura 3.4* muestra la estructura del módulo a través de un diagrama de clases.

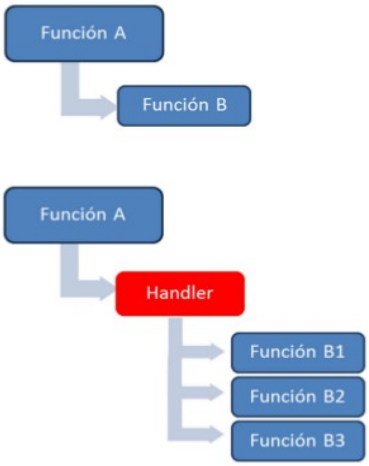


Figura 3.3. Diferencia en el flujo de ejecución sin hacer uso de handlers (parte superior) y usándolos (parte inferior)

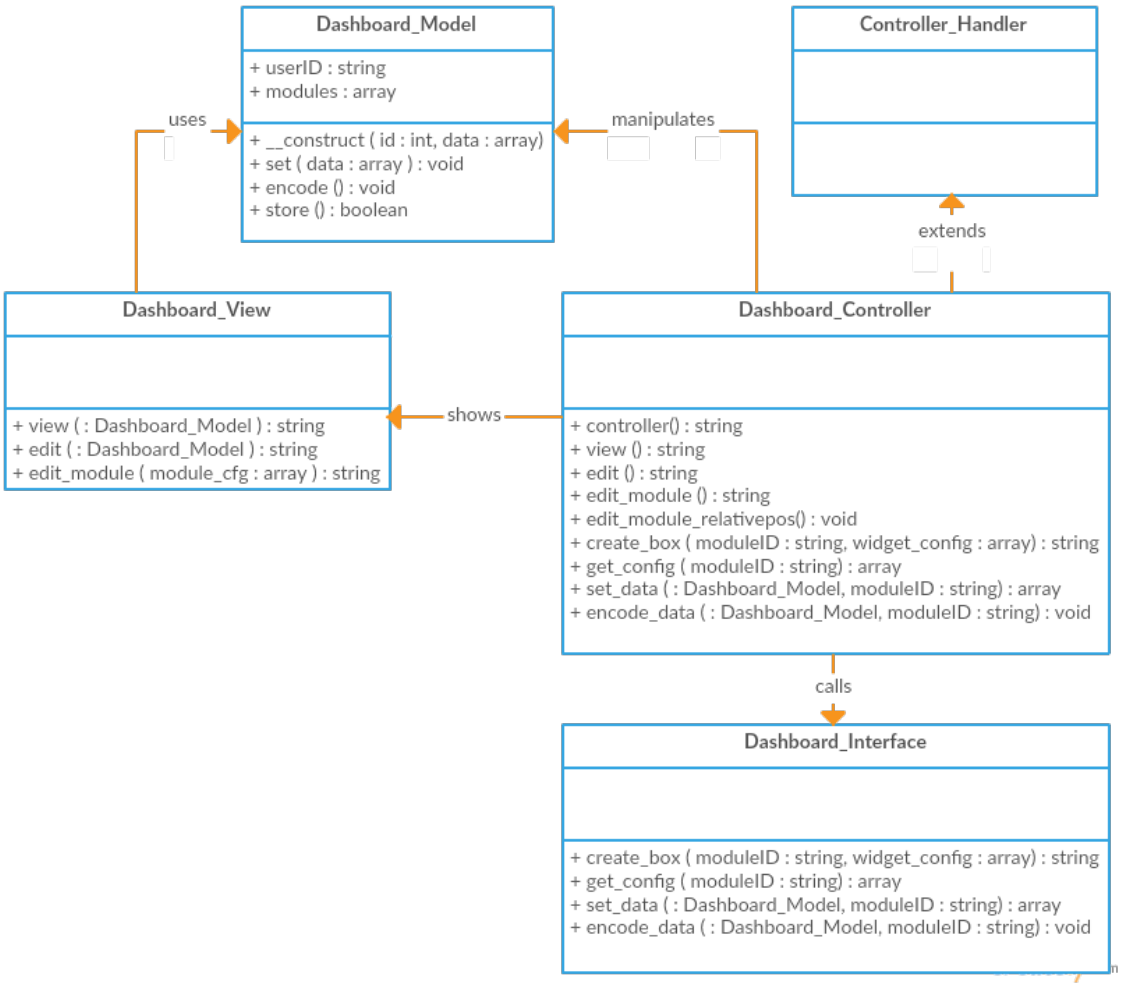


Figura 3.4. Diagrama de clases del módulo

En la estructura que se muestra en este diagrama, se definen las siguientes clases:

- *Controller_Handler*: es una clase que se usa en los módulos del sistema que necesitan hacer uso de los *handlers*. Sus atributos y métodos sirven de apoyo a la gestión de los *handlers* desde la clase hija y no suponen ninguna implicación adicional en el desarrollo del módulo, por lo que se han omitido en el diagrama.
- *Dashboard_Controller*: controlador (que extiende de la clase anteriormente mencionada) del *MVC*.
- *Dashboard_View*: vista del *MVC*.
- *Dashboard_Model*: modelo lógico del *MVC*.
- *Dashboard_Interface*: es la interfaz que define las operaciones necesarias para que un módulo se pueda monitorizar en el panel de control.

4. Implementación

En el presente capítulo se explican los aspectos más importantes que afectan a la fase de implementación del proyecto, es decir, el entorno en el que se ha desarrollado y cómo se cumplen los requisitos definidos anteriormente.

4.1. Entorno de desarrollo

El proyecto se ha desarrollado haciendo uso del entorno de desarrollo de *Netbeans* [12]. Se trata de un *IDE* libre, de código abierto, con soporte para múltiples lenguajes de programación y con una comunidad internacional de usuarios y desarrolladores. Además, tiene la capacidad de autoformatear el código de acuerdo con las preferencias del desarrollador, cosa que resulta muy útil para mantener todo el código siguiendo los mismos estilos. Por otro lado, también permite introducir código con atajos de teclado, lo que puede ahorrar tiempo y evitar errores en algunas ocasiones.

El principal lenguaje de programación utilizado en la empresa, y también en este proyecto, es *PHP*. Este lenguaje es muy conocido para desarrollo de aplicaciones Web en el lado del servidor, con un repertorio amplio de funciones que se han aprovechado en este TFG. *HTML* es el lenguaje que se emplea para el desarrollo de la interfaz de usuario (o cliente), con el fin de dar estructura y contenido a la aplicación Web. Igualmente se utiliza *JavaScript* (y tecnologías *AJAX* usando la librería *jQuery*) para la carga de datos asíncronamente y actualización de elementos *DOM* (*Document Object Model*) de los interfaces de las diferentes pantallas del panel de control.

Por último, en el entorno de desarrollo se ha utilizado el sistema de control de versiones *Git* [13], debido a que varios desarrolladores trabajan en el sistema simultáneamente. Por la misma razón, se ha empleado una metodología común a todos los desarrolladores de la empresa en términos de documentación, nomenclatura de variables y funciones, etc.

4.2. Dashboard

Como todos los módulos del sistema, el *dashboard* se divide en tres partes:

- Modelo lógico, que representa una configuración del panel de control para un usuario concreto, es decir, cada instancia de *Dashboard_Model* tendrá un identificador de usuario y un *array* que almacene, para cada módulo monitorizable, los parámetros que este usuario ha configurado. Este modelo lógico se puede considerar una “copia” de un documento de la base de datos, cuyo esquema se explican en el anexo II.
- Vista, que genera el código *HTML* para darle formato a las diferentes páginas de la aplicación Web.

- Controlador, al que llegarán las peticiones del cliente. Estas peticiones pueden ser de tres tipos, dependiendo de un parámetro que indica el tipo de acción: 1) *edit*, para mostrar la pantalla de configuración; 2) *edit_module*, para mostrar la caja de configuración de un módulo en concreto (especificado en otro parámetro); y 3) *view* para la pantalla de visualización.

Las soluciones propuestas para satisfacer los requisitos funcionales relacionados con la pantalla de configuración se han implementado de la siguiente forma:

- **RF2**: al usuario se le mostrará un *checkbox* múltiple en el que pueda elegir los módulos que quiere configurar. De esta manera, cada vez que active el *checkbox* de un módulo, se carga la caja de configuración de ese módulo. Como es de esperar, al desactivarlo se oculta dicha caja. La carga de estas cajas se hace mediante *AJAX*, realizando una petición asíncrona al controlador del módulo, con la acción *edit_module* y cargando el resultado en un *div*.
- **RF3** y **RF4**: en todas las cajas de configuración de los módulos se mostrarán los parámetros de configuración comunes y los específicos de cada módulo. La *figura 4.1* muestra la sencillez con la que se resuelve este problema gracias al uso de *handlers*.

Figura 4.1. Flujo de ejecución cuando se muestra la caja de configuración de un módulo

Figura 4.2. Flujo de ejecución cuando se muestra un widget

Por otro lado, las soluciones propuestas para satisfacer los requerimientos de la pantalla de visualización del panel de control se han implementado de la siguiente forma:

- **RF1**: por cada *widget* se ha inicializado una *poll* que realiza peticiones periódicas, según el tiempo de refresco que tenga asignado, al controlador del módulo del *widget*. De esta forma, se ha implementado en cada controlador de estos módulos una función que se llama cuando recibe estas peticiones. Esta función tiene como objetivo devolver los datos del módulo actualizados, en formato *JSON*, de tal manera que en el lado del cliente se procesarán dichos datos y se actualizará el *widget*. Esto puede llevar a generar una sobrecarga tanto del navegador web

como del servidor, debido a que el usuario dispone de un menú en la parte superior, por el que puede navegar hasta las distintas partes del sistema web usando *AJAX* (de forma asíncrona), lo que provoca que las *polls* queden activas hasta que se recarga la web síncronamente. Si a este hecho se suma que el usuario cargue asíncronamente, usando este menú, más de una vez la pantalla de visualización del panel de control, se crean varias *polls* que, no sólo sobrecargan el sistema, sino que provoca que **no se respete el tiempo de actualización** configurado. Para resolver este problema se asigna un **identificador aleatorio y único al *dashboard*** cada vez que se visualice. De esta forma una *poll* estará ligada a ese identificador y dejará de tener efecto si ya no existe un elemento con ese identificador.

- **RF5**: en cada *widget* existe un botón que esconde o muestra el contenido del mismo, dependiendo de su estado anterior (si ya estaba mostrado o estaba oculto).
- **RF6**: la librería *Sortable* [14] permite que se pueda hacer *drag and drop* sobre los elementos contenidos en un *div*. Por lo tanto sólo es necesario incluir todos los *widgets* dentro de un mismo *div*.
- **RF7**: la librería *Sortable* también permite asociar una función al evento *drag and drop* para que se dispare cuando ocurra. De esta forma, cada vez que se cambia de posición un *widget* se hará una solicitud *AJAX* al servidor. En esta solicitud se envían las posiciones absolutas de todos los *widgets*. El servidor entonces actualizará en la base de datos el nuevo orden de los *widgets* del usuario.

4.3. Handlers

Para implementar los *handlers* se ha creado una clase por cada uno de los distintos módulos que se monitorizarán en el *dashboard*. El nombre de esta clase será *Módulo_Dashboard* (por ejemplo, *EstacionesMeteo_Dashboard*) y, por consiguiente, el nombre del fichero que la contiene se denominará *módulo_dashboard.php*. La razón por la que se separa esta funcionalidad en otra clase, en vez de introducirlo en el controlador, es porque así el *autoloader* del sistema evitará cargar código innecesario cuando se acceda al módulo como tal. La *figura 4.3* explica la diferencia entre hacerlo de una manera u otra.

Estas clases específicas para cada módulo tienen la función de generar las distintas representaciones de datos (tablas, gráficas, mapas...) e inicializar las *polls* que realizarán periódicamente peticiones asíncronas al módulo concreto. Así podrá obtener los datos más recientes y actualizarlos en el navegador del usuario.

Para más información sobre el cumplimiento de los requisitos relacionados con los *widgets*, ver anexos III y IV.

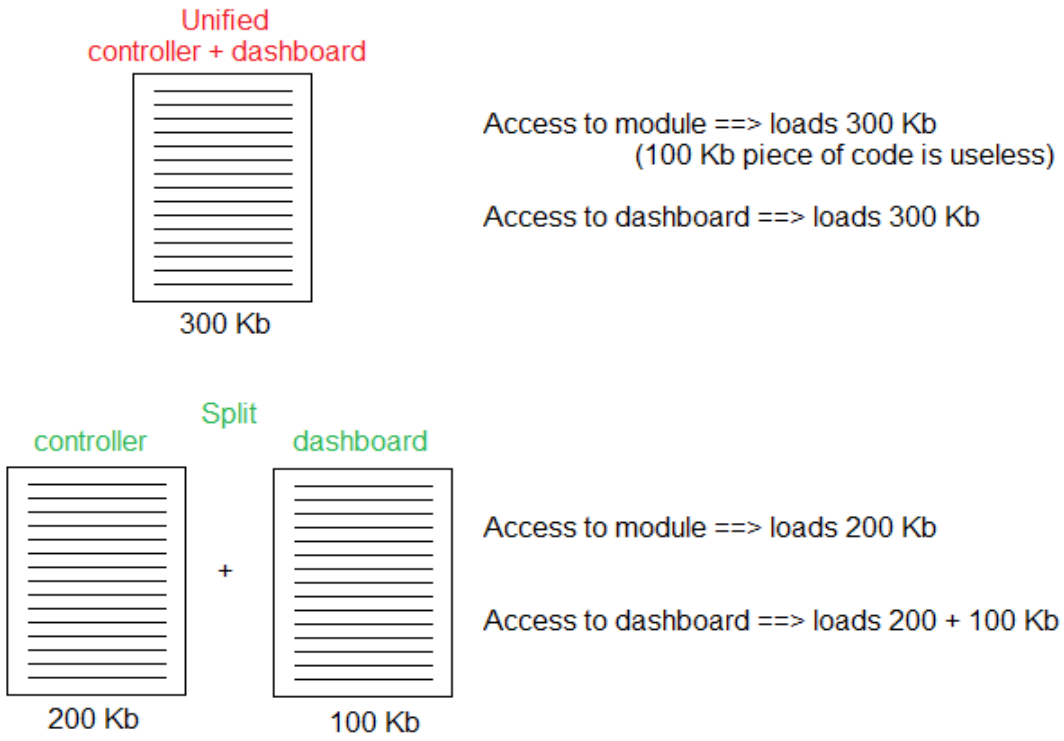


Figura 4.3. Comparación en el comportamiento del Autoloader

4.4. Cumplimiento de otros requisitos

El cumplimiento de los requisitos no funcionales descritos en la fase de análisis se sostiene en los siguientes puntos:

- **RNF1**: el dinamismo creado gracias al uso frecuente de *JavaScript*, la existencia de pequeñas ayudas y el fácil uso del módulo en general satisfacen este requisito.
- **RNF2**: el uso de la librería *Bootstrap* [15] en ambas pantallas del *dashboard* se permite el acceso y uso desde diversos dispositivos (tanto móviles, *tablets*, pantallas medianas o los grandes paneles de los centros de gestión). *Bootstrap* divide la pantalla en doce columnas, de manera que, asignando unas clases específicas a los elementos *HTML*, se les asigna a su vez un número de columnas de la pantalla. Por ejemplo, si un elemento *HTML* tiene las clases *col-xs-12* y *col-md-4* entonces en pantallas pequeñas (dispositivos móviles principalmente) ocupará la pantalla completa y en pantallas medianas (como monitores de 19 pulgadas) un tercio de la pantalla.
- **RNF3**: para añadir un nuevo módulo al *dashboard* sólo es necesario crear una nueva clase que implemente la interfaz *Dashboard_Interface* y añadir el identificador del módulo al fichero de configuración donde se indican los módulos que están disponibles para ser monitorizados.
- **RNF4**: se han separado las funcionalidades en distintos bloques de

código, de esta forma algunas de las funcionalidades desarrolladas específicamente para un módulo se pueden reusar en futuros módulos.

- **RF12**: se ha hecho un análisis exhaustivo de diversas librerías orientadas a satisfacer este requisito. Se ha llegado a la conclusión de que la mejor opción es la librería *Slick* [16], que sirve para crear *scrolls* horizontales con múltiples opciones para configurarlos al completo.
- **RF13**: se ha modificado el módulo del *core Google Charts* para añadir la posibilidad de actualizar las gráficas por *AJAX*.
- **RF14**: al igual que en el requisito RF12, se ha realizado un análisis de algunas herramientas que puedan solventar este requisito. Se ha determinado que la mejor herramienta es la librería *Sortable*, que al igual que *Slick*, resulta muy sencilla de usar.

5. Tests Unitarios

Los tests unitarios son una forma de comprobar el correcto funcionamiento y comportamiento del código, una vez terminada la fase de desarrollo. Estas pruebas facilitan el descubrimiento de errores y su posterior corrección antes de integrar los componentes desarrollados en el sistema. Es por ello que este tipo de tests suele **automatizarse** a través de herramientas de integración contínua, como *Jenkins* [17]. Los tests unitarios del *dashboard* no sólo serán útiles antes de introducirlo en el sistema, sino que también servirán para detectar errores en posibles modificaciones futuras que se hagan sobre él, como por ejemplo una actualización en los atributos de cada módulo visualizado en el dashboard.

Para verificar el correcto funcionamiento del *dashboard*, se han empleado los *frameworks PHPUnit* [18] y *Selenium* [19].

El resto de tests (de usuarios, de integración, etc) los lleva a cabo otro desarrollador de Iternova (acordado previamente con la empresa).

5.1. PHPUnit

PHPUnit es un *framework* que permite programar pruebas unitarias para *PHP*, con el objetivo de detectar y corregir errores lo antes posible. Para ello utiliza el concepto de **aserciones**, es decir, una comprobación lógica (verdadero o falso) que verifica que el resultado de la ejecución de un bloque de código es el esperado. El fallo de una aserción implica el lanzamiento de una excepción y, por consecuencia, se abandona la ejecución de las pruebas unitarias.

La mayoría de las pruebas unitarias realizadas en este módulo y, en general, en *PHP* tienen que ver con los **tipos** de resultados devueltos por las funciones, es decir, gran parte de estas pruebas consiste en verificar que el resultado de una función es del tipo *array*, *string*, *integer*, *float*, etc. Esto es debido a que *PHP* es un lenguaje de programación débilmente tipado (o no tipado), lo que significa que hay muy pocas restricciones (o ninguna) en cuanto a los tipos de variables. Por lo tanto, una variable en un momento determinado puede valer "PRUEBA" (de tipo *string*) y en otro momento se le puede ser asignado el valor 5 (*integer*). Además, en las funciones no se define el tipo de dato que se devuelve. En ocasiones, esto puede provocar fallos durante la ejecución de alguna parte del código, ya que una función podría estar usando de manera incorrecta un dato devuelto por otra función (por ejemplo usando un *integer* como *float*), acabando en un resultado de error. Es por ello que algunas de las pruebas unitarias diseñadas en este módulo establecen las **restricciones** que el lenguaje no tiene (lo cual no quiere decir que sea una desventaja de *PHP*) en la fase de testeo.

El objetivo final, junto con la documentación tanto en el propio código como en la *wiki*, es evitar errores de esta naturaleza.

5.2. Selenium

Selenium es un *framework* orientado al desarrollo de pruebas que verifiquen el correcto funcionamiento de las **interfaces de usuario** en el navegador web. Esto lo consigue ejecutando una instancia de un navegador web (tiene soporte para *Firefox*, *Chrome*, *Internet Explorer*, *Safari*, *Opera* y navegadores para móviles) y ejecutando las pruebas que se han programado previamente. Estas pruebas pueden programarse en varios lenguajes: *Java*, *C#*, *Ruby*, *Python* y *Node.js* (*JavaScript*).

El uso de *Selenium* ayuda a detectar fallos, por ejemplo, en el funcionamiento de efectos dinámicos o en la carga de elementos por *AJAX*. Permite, entre otras cosas, situar el puntero del ratón en una posición determinada, o hacer *click* en un elemento *HTML* concreto. Algunas de las pruebas que se han realizado son las siguientes:

- Comprobar el correcto funcionamiento del *drag and drop* de *widgets*. Para ello se arrastra el primer *widget* que aparece en la pantalla a la posición del segundo, simulando el movimiento de ratón de un humano. Después, se comprueba, mediante los elementos *HTML*, que el nuevo orden de *widgets* es el que debería ser.
- Verificar el funcionamiento del botón de ocultar/mostrar los *widgets*. En este caso se comprueba, para todos los *widgets*, haciendo *click* sobre el botón correspondiente, si los módulos se esconden y se muestran debidamente, de nuevo, accediendo a las propiedades *HTML* de los elementos que componen el módulo.
- Revisar si los *checkbox* de la pantalla de configuración cargan y ocultan las cajas de configuración de los módulos. Esto se consigue accediendo, después de hacer *click* sobre los *checkbox*, a las propiedades *HTML* del elemento que indican si el módulo está oculto o mostrado.

6. Resultados

En este apartado se resumen los resultados obtenidos al finalizar el desarrollo y testeo del *dashboard*.

6.1. Configuración y visualización del dashboard

Se han cumplido todos los requisitos relacionados con la configuración del *dashboard*. En en el anexo III se muestran pantallas de ejemplo de configuración de un panel de control. En la primera se observa cómo se escogen los parámetros de cada módulo a configurar, mientras que en la segunda se ve el *feedback* (retroalimentación) recibido.

También se han conseguido satisfacer los requisitos relacionados con la visualización del panel de control. En las figuras del anexo IV se observa el resultado final de la pantalla de visualización. En las dos primeras están todos los módulos mostrados (se pueden apreciar las distintas representaciones de datos usadas), mientras que en la tercera se observa cómo, al esconder algunos módulos, los demás se ajustan para no dejar espacio libre sin usar en el *grid*.

6.2. Otros

Además de cumplir con los anteriores requisitos, se ha considerado apropiado crear un apartado en el manual de ayuda para el usuario con las instrucciones de uso del *dashboard*, que se puede apreciar en el anexo V.

De cara al desarrollador, se han documentado las características básicas del módulo en la *wiki*, que se describen brevemente en el anexo VI.

6.3. Integración y puesta en producción

Tras pasar con éxito los tests correspondientes, el panel de control se ha instalado en el sistema *SmartRoads* y ya está siendo usado por aquellos usuarios finales que lo necesitan.

7. Conclusiones

7.1. Sobre el módulo desarrollado

El módulo desarrollado (que ya se encuentra en funcionamiento) significa la introducción de una herramienta de gran utilidad que producirá una mejora importante en las tareas de los gestores de carreteras, ya que permite tener una visión detallada del estado de todo el sistema. Esto agilizará el análisis de los datos mostrados en los módulos monitorizados y, por consecuencia, la toma de decisiones.

Así mismo, se prevé la introducción a corto plazo de otros módulos en el *dashboard*, como por ejemplo el de accidentalidad y tramos de concentración de accidentes.

Además, debido a diversos factores como la documentación y las técnicas y patrones usados en la implementación, resulta muy fácil de añadir nuevos módulos para ser monitorizados y el código del panel de control cumple con el requisito de mantenibilidad, por lo que de cara al desarrollador satisface los objetivos.

Por último, de cara al usuario final se destaca la usabilidad del módulo, tanto en la correcta visualización en diversos dispositivos (*responsive*) como en la facilidad de uso del panel de control gracias a las ayudas existentes no sólo en el manual de usuario sino también en la propia interfaz.

7.2. Personales y profesionales

Gracias al desarrollo de las prácticas (ver breve descripción en el anexo VIII) y de este TFG en la entidad Iternova S.L. he adquirido experiencia de gran valor en diversos ámbitos:

- En lo tecnológico, dado que he conocido y usado nuevas tecnologías relacionadas con el mundo del desarrollo web como, por ejemplo, *Jenkins* (integración continua), *Ant* (para automatización de tareas), *Gearman* (colas de procesado), *MongoDB* (base de datos no relacional) y *PHP* (programación en el lado del servidor).
- En lo que respecta a la gestión de proyectos, pues he utilizado herramientas que desconocía, y que facilitan tareas en este ámbito. Por ejemplo: *Trello* (para la gestión de tareas), *Bugzilla* (para el control de esfuerzos) y *DokuWiki* (para la documentación, por ejemplo, de cómo crear y configurar un entorno de desarrollo o de problemas frecuentes a la hora de desarrollar).
- En relación al trabajo en equipo, dado que es importante usar el mismo *framework* que los demás desarrolladores, crear y consultar documentación cuando resulte necesario.

- En lo laboral, ya que, me debo de imponer una rutina de trabajo para ser lo más productivo posible y cumplir todas las tareas en su plazo.

Por último, señalar que las horas de esfuerzo totales dedicadas a este TFG se indican en el anexo VII. No obstante, señalar que para el control de esfuerzos no se ha tenido en cuenta el tiempo de elaboración de esta memoria.

8. Bibliografía

[1] «MySQL». https://www.mysql.com/
[2] «MongoDB». https://www.mongodb.org/
[3] «PHP». http://php.net/
[4] «jQuery». https://jquery.com/
[5] «Google Maps API». https://developers.google.com/maps/
[6] «Google Charts API». https://developers.google.com/chart/
[7] «JSON». http://www.json.org/
[8] «BSON». http://bsonspec.org/
[9] «Gearman». http://gearman.org/
[10] «Cron». https://en.wikipedia.org/wiki/Cron
[11] «Shapeshift». http://mcpants.github.io/jquery.shapeshift/
[12] «NetBeans». https://netbeans.org/
[13] «Git». https://git-scm.com/
[14] «Sortable». http://rubaxa.github.io/Sortable/
[15] «Bootstrap». http://getbootstrap.com/
[16] «Slick». http://kenwheeler.github.io/slick/
[17] «Jenkins». https://jenkins-ci.org/
[18] «PHPUnit». https://phpunit.de/
[19] «Selenium». http://www.seleniumhq.org/

Anexo I. Funcionamiento de Gearman

Gearman proporciona tres componentes principales al desarrollador, que son:

- *Job Server*: es la parte que se encarga de recibir *jobs* (trabajos) por parte de los clientes y asignarlos a un servidor.
- *API* del cliente para crear *jobs* y enviarlos a un *job server*.
- *API* del *worker* para ejecutar *jobs* y devolver el resultado.

El funcionamiento de una aplicación con estos tres componentes es sencillo. Para explicarlo, se pondrá un ejemplo típico donde el servidor web tiene que realizar determinados trabajos que son muy costosos en tiempo de cómputo, por lo que necesita que otros servidores especializados en esto hagan estos trabajos. A este escenario hay que añadirle los servidores que hacen de intermediarios, los *job servers*. Estos servidores reciben suscripciones de los servidores *workers*, indicando los trabajos que son capaces de realizar. Una vez estén suscritos dichos trabajos asociados a los *workers* en el *job server*, el servidor web ya puede empezar a enviar trabajos a un *job server* para que encuentre a un *worker* que no esté ocupado y sea capaz de realizar dicho trabajo. Una vez el *worker* ha terminado el trabajo, envía el resultado al *job server* que a su vez devuelve el resultado al servidor web, que inicialmente había solicitado la ejecución del trabajo. El desarrollo de este escenario se entiende mejor en el diagrama de secuencia de la *figura 9.1*. Para simplificar el diagrama se ha omitido la parte de inicialización de los servidores, y sólo se han tenido en cuenta un *job server* y un *worker*.

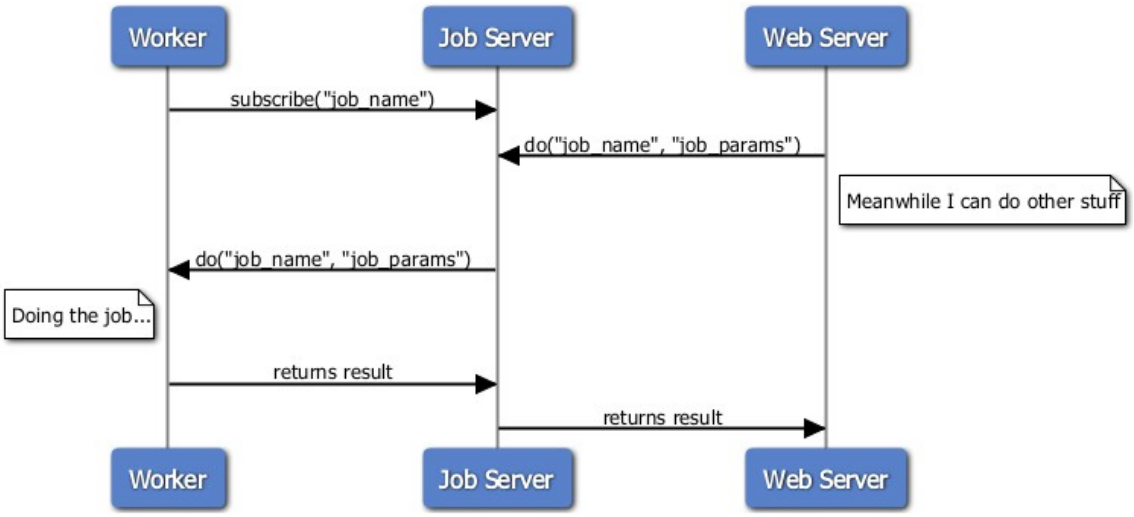


Figura 9.1. Funcionamiento de Gearman

Anexo II. Esquema de los documentos de la base de datos

Al usar una base de datos documental, como es MongoDB, no es necesario que todos los documentos de una colección se ajusten a un mismo esquema. No obstante, en este caso todos los documentos que guardan la configuración del *dashboard* de cada usuario necesitan seguir el mismo patrón, ya que todos necesitan los mismos atributos y no hay ninguno opcional. Esos atributos son, obviando el identificador MongoID, los siguientes:

- *UserID*: usuario al que pertenece esta configuración de *dashboard*.
- *Modules*: conjunto de módulos configurados por el usuario, que a su vez contiene los siguientes atributos:
  - *Active*: indica si el módulo se mostrará en el *dashboard* del usuario.
  - *Polltime*: tiempo de refresco de la información del módulo.
  - *Setup*: modo de mostrar los datos del módulo. Esto puede ser por ejemplo en tabla, en gráfico, en mapa de Google, o una combinación de los anteriores.
  - *Relativepos*: guarda el módulo que se encuentra anterior a éste, es decir, una posición relativa. El primer módulo tendrá una posición relativa *first_item*.
  - *Specific*: contiene atributos de configuración específicos del módulo. Se delega la gestión de este atributo en el *handler* correspondiente.

En la *figura 9.2.* se muestra un ejemplo de documento de esta colección.

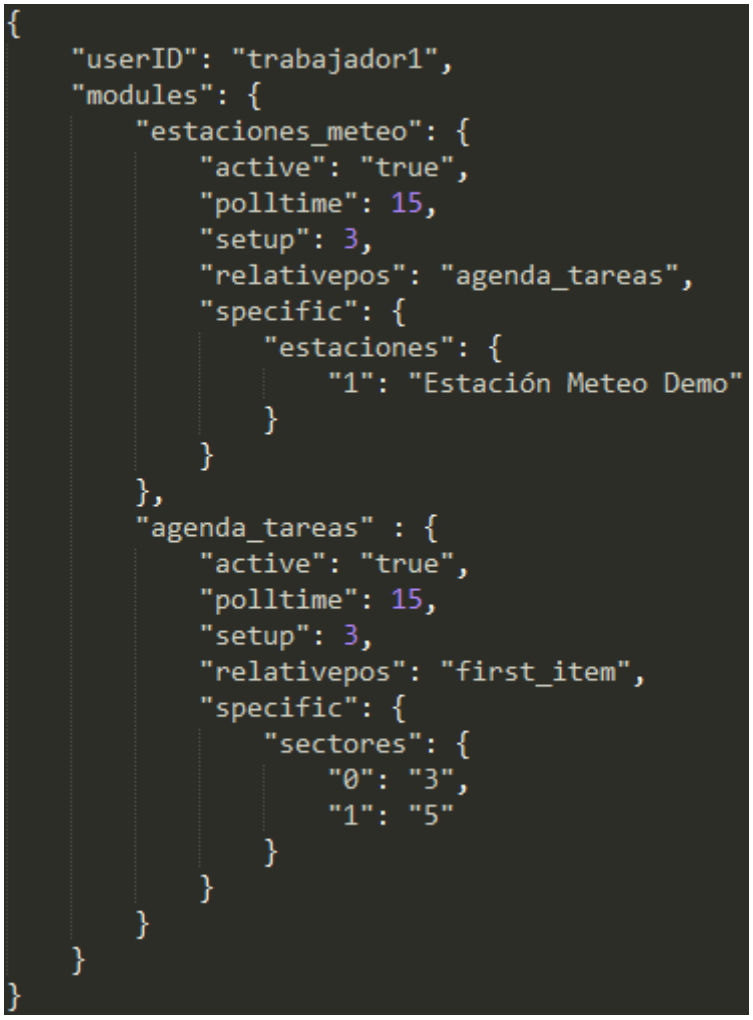


Figura 9.2. Ejemplo de documento de la base de datos

En este documento se observa que el usuario 'trabajador1' ha configurado los dos módulos activos en el *dashboard* de manera que se visualicen los dos (atributo a*ctive*) con una frecuencia de actualización de 15 segundos *(polltime).* El primer módulo que aparecerá será el de Agenda de Tareas, ya que tiene el atributo r*elativepos* a 'first_item', mientras que el segundo será el de Estaciones Meteorológicas. Ambos tienen una s*etup* de '3', lo que significa que el módulo de Estaciones Meteorológicas mostrará los datos en forma de tabla y gráfica, mientras que el de Agenda de Tareas también lo hará en forma de mapa. Los atributos específicos de cada módulo indican que en el de Estaciones Meteorológicas se mostrará información sólo de la estación con identificador 1, que se denomina Estación Meteo Demo, mientras que en el de Agenda de Tareas se mostrará información de los sectores 3 y 5. Como se puede observar, el tratamiento/procesamiento del atributo s*pecific* es independiente de cada módulo.

Anexo III. Pantallas de configuración del panel de control

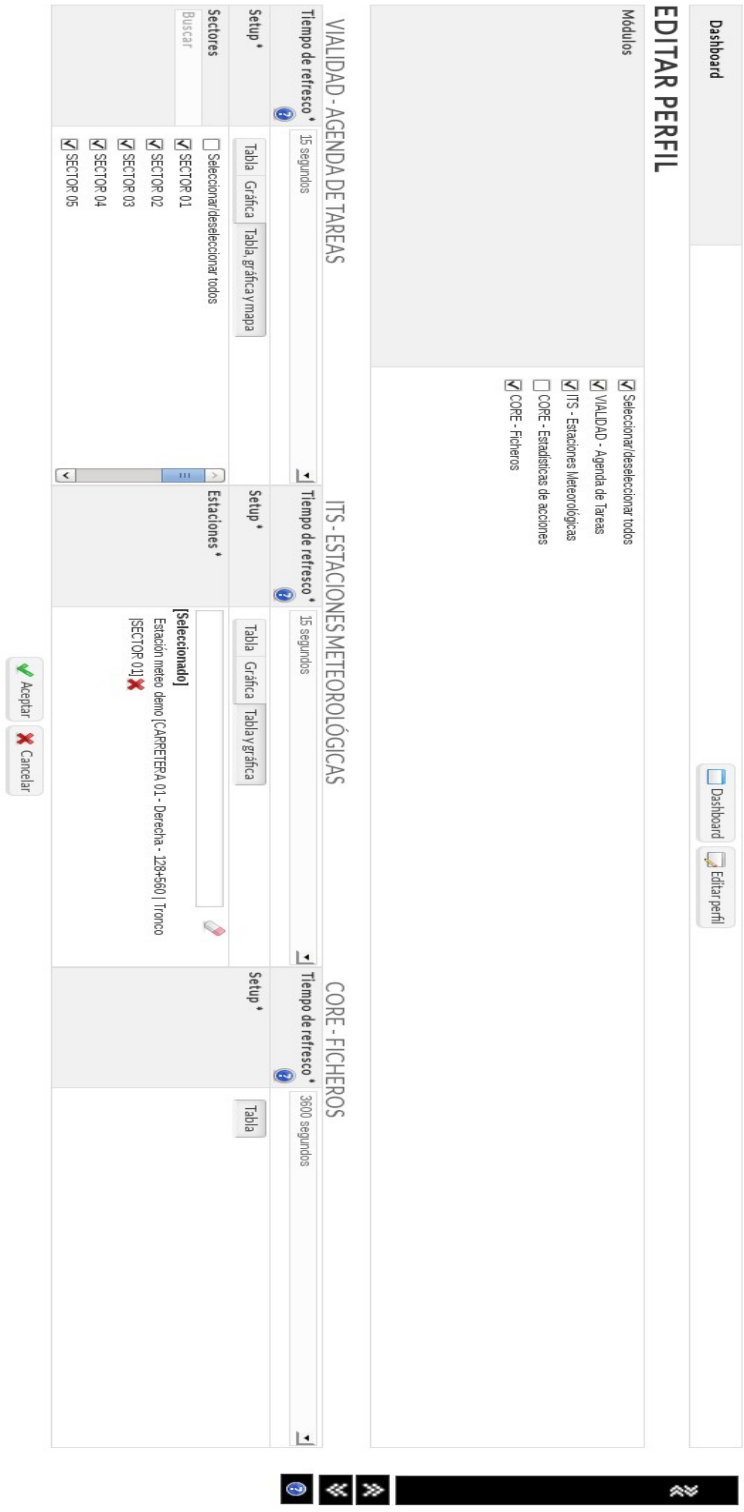


Figura 9.3. Configuración del panel de control

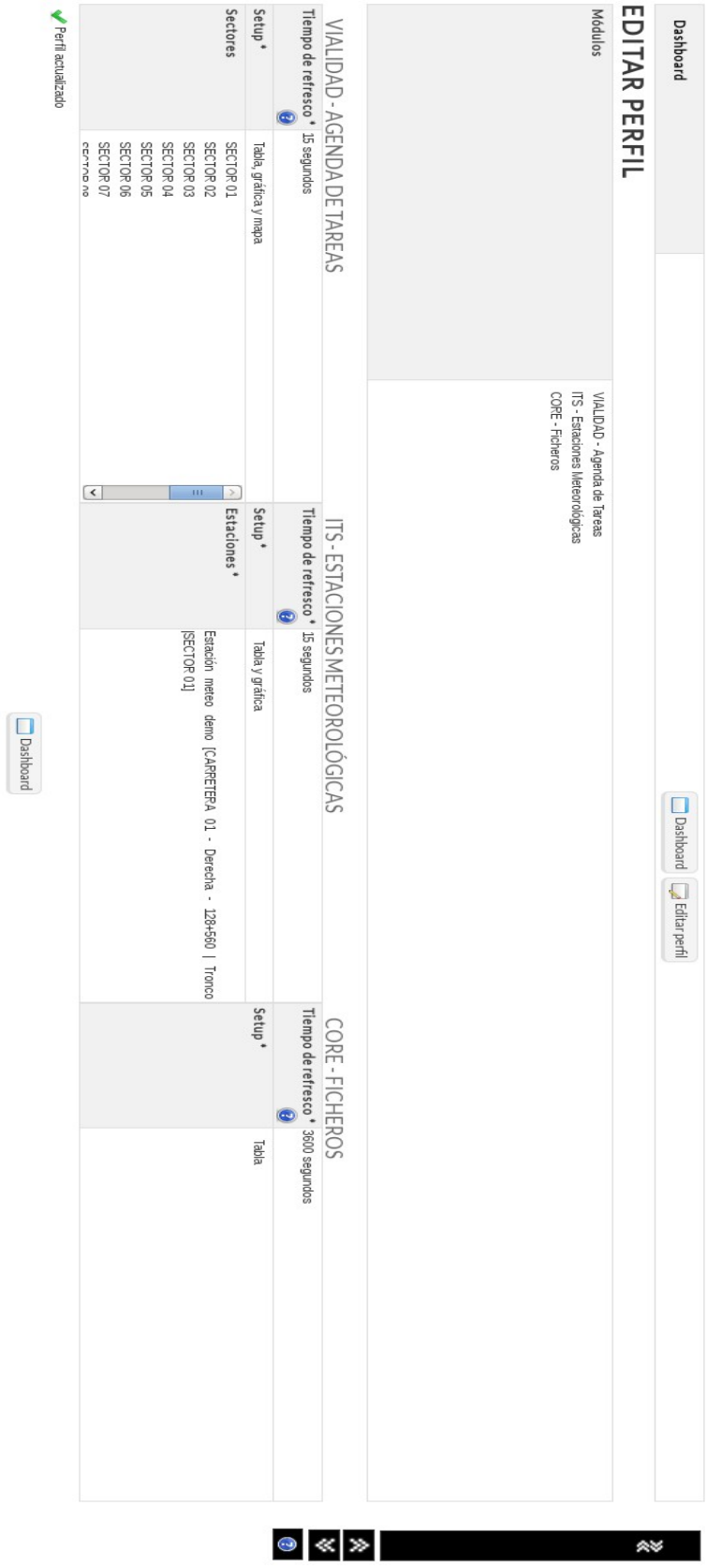


Figura 9.4. Retroalimentación al configurar el panel de control

Anexo IV. Pantallas de visualización del panel de control

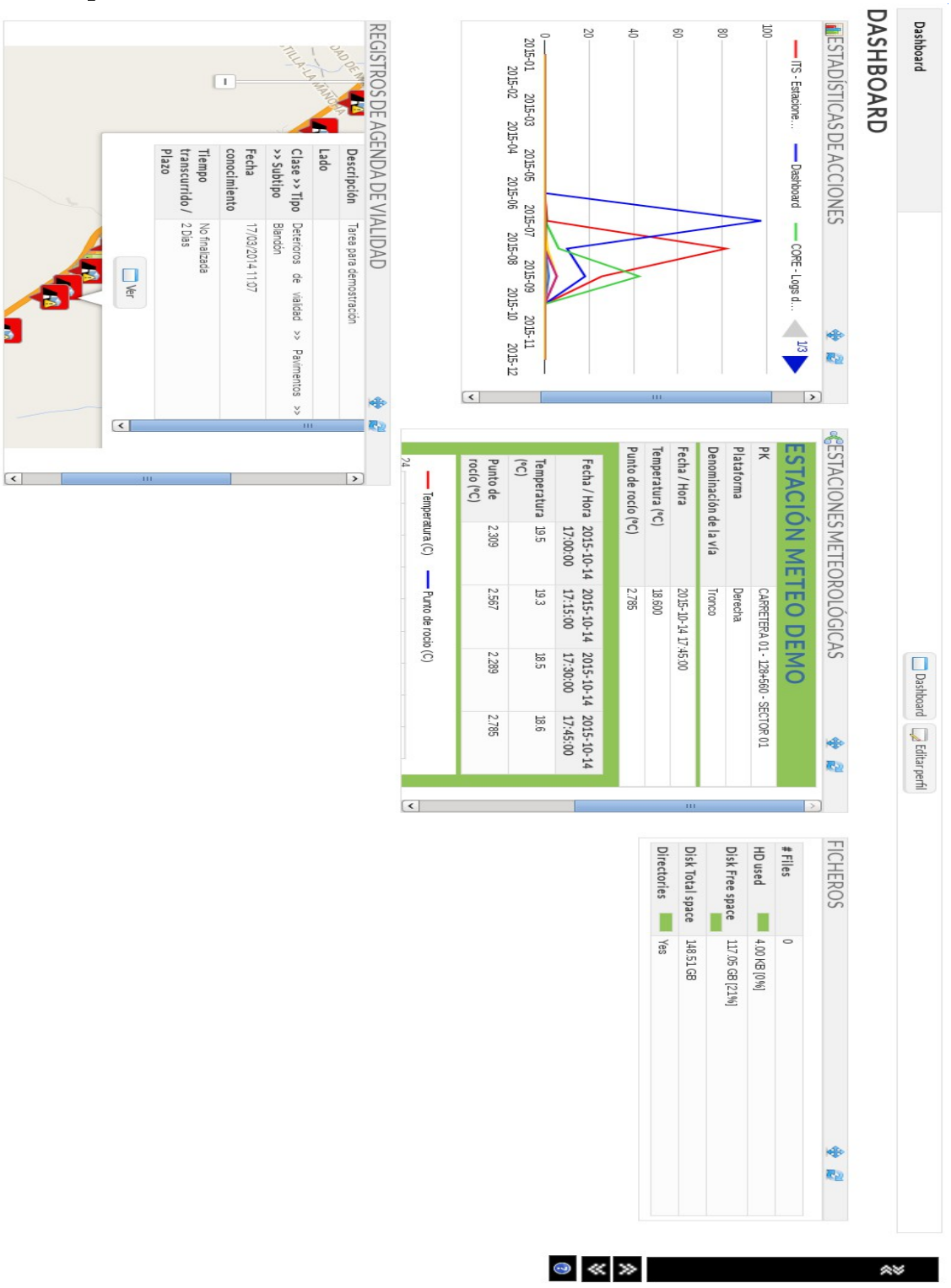


Figura 9.5. Visualización del panel de control (I)

Figura 9.6. Visualización del panel de control (II)

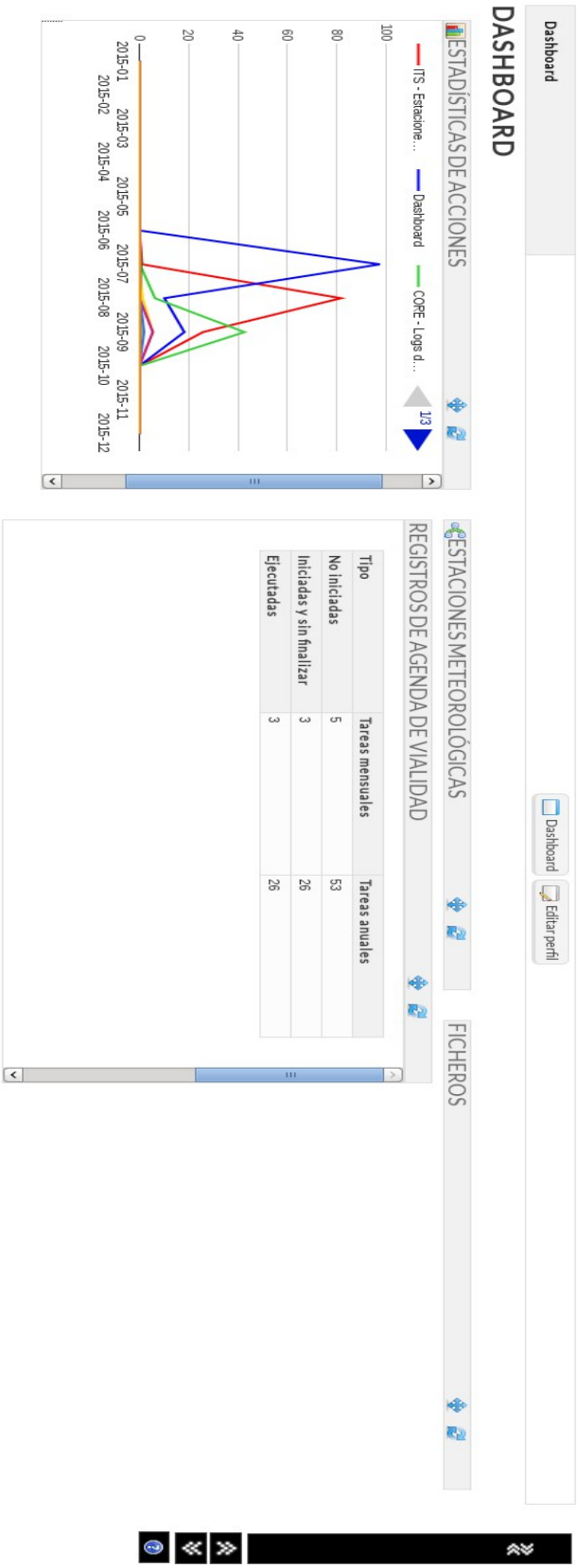


Figura 9.7. Visualización del panel de control (III)

Anexo V. Manual de ayuda para el usuario

Para facilitar el uso del *dashboard* por parte del usuario final, se han creado botones de ayuda en algunos campos de la configuración del *dashboard* (en este caso existe una ayuda en el campo de "Tiempo de Refresco"), tal y como se puede apreciar en las figuras del anexo I.

Además, se han añadido unas instrucciones de uso del panel de control que se pueden ver haciendo *click* en el botón de ayuda que aparece en la parte lateral derecha (símbolo de interrogación en un círculo con fondo azul). Estas instrucciones de uso se añaden al manual de usuario, en caso de que éste desee descargarlo, junto con las instrucciones de los demás módulos activados en el sistema. En la *figura 9.8* se muestra esta ayuda que puede consultar y descargar el usuario.

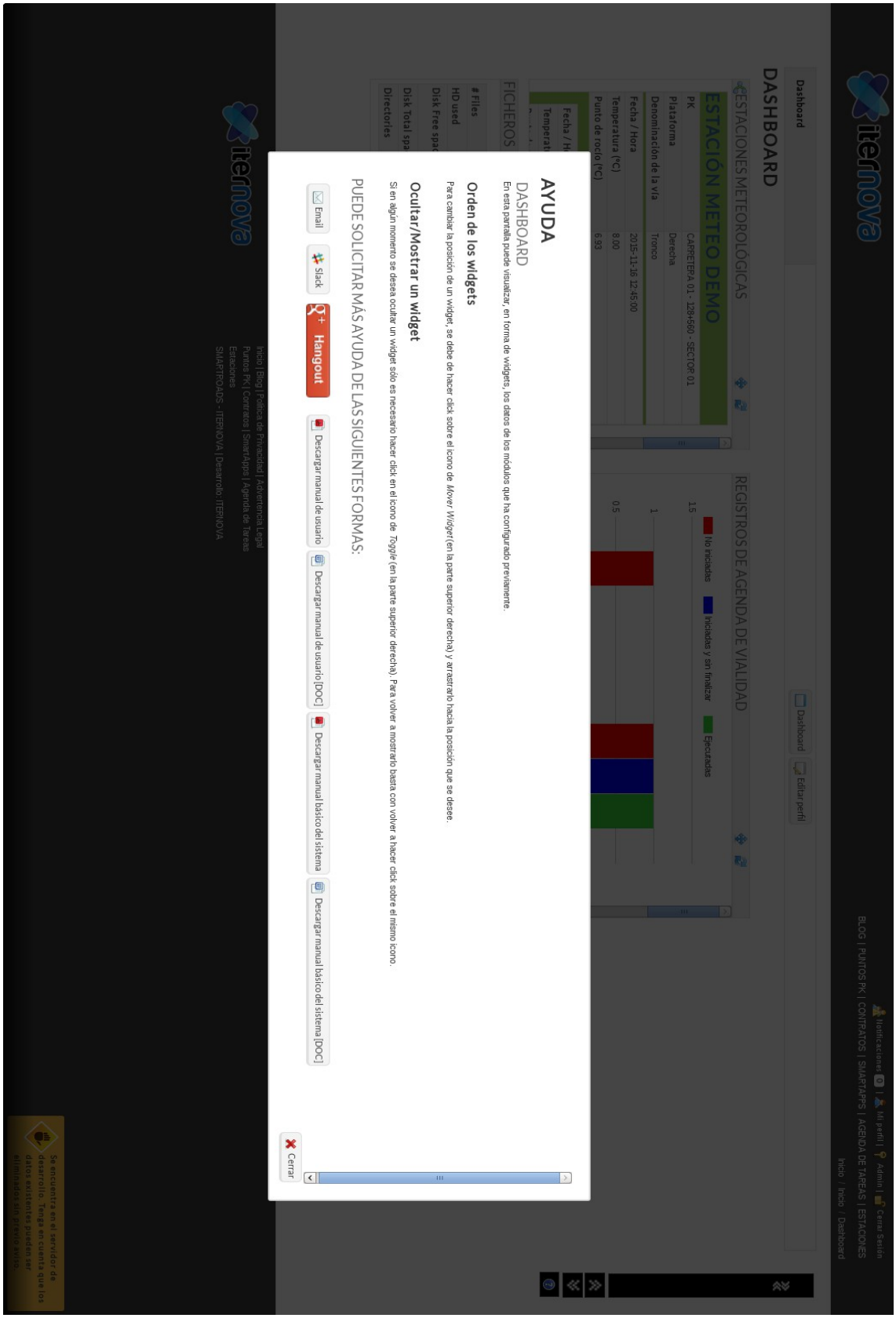


Figura 9.8. Manual de ayuda para el usuario

Anexo VI. Documentación para el desarrollador

Se ha creado una página con documentación para los desarrolladores en la *wiki* de la empresa. Esta página contiene información de interés sobre el funcionamiento del módulo.

En la *figura 9.9* se puede apreciar una porción de esta página. En la parte lateral derecha se muestra una tabla para acceder de forma rápida a los contenidos que se desean consultar. Estos contenidos son, principalmente, datos generales del módulo (autor, descripción, versión...), funcionamiento de los métodos principales del controlador (qué parámetros reciben, qué devuelven, etc) y el *changelog*.

Figura 9.9. Documentación para el desarrollador

Anexo VII. Recopilación de esfuerzos

La *tabla 9.1* contiene los códigos usados en la hoja de recopilación de esfuerzos:

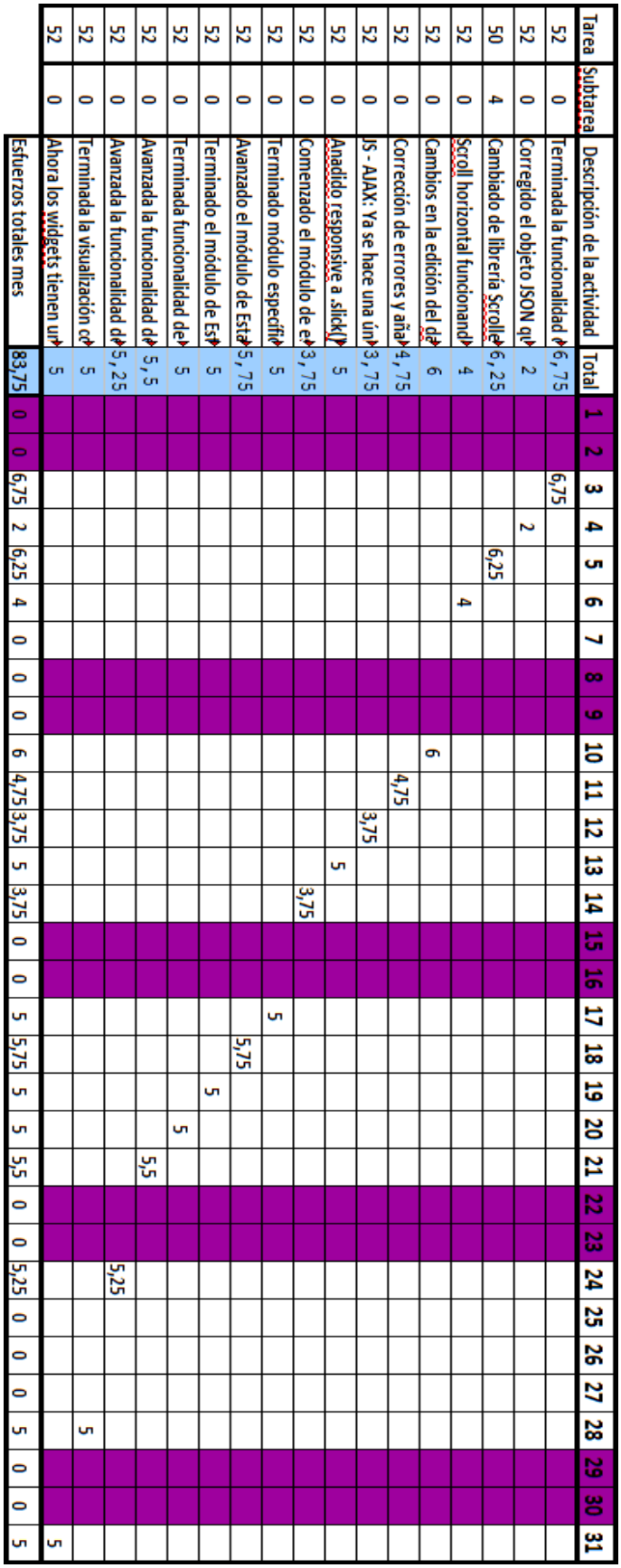
Código	Descripción
50	Análisis
50.1	Requisitos funcionales
50.2	Librerías alternativas a Google Charts
50.3	Librerías alternativas a jQuery.Shapeshift
50.4	Librería para hacer scroll horizontal
51	Diseño
52	Implementación
53	Pruebas unitarias
53.1	Pruebas con PHPUnit
53.2	Pruebas con Selenium
54	Documentación/Manuales
54.1	Manual de ayuda para el usuario
54.2	Documentación para desarrolladores
55	Preparación entorno de desarrollo
55.1	Instalación/Configuración de máquina virtual
55.2	Configuración de NetBeans

Tabla 9.1. Códigos de la hoja de recopilación de esfuerzos

Las *tablas 9.2*, *9.3* y *9.4* contienen la cantidad de horas invertidas por día y por tarea a lo largo de la realización del TFG. La *tabla 9.5* contiene un resumen de las anteriores.

Tarea	Subtarea	Descripción de la actividad	Total	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
55	1	Instalación de la VM Vagran	6	6																														
55	1	Configuración de la VM	4,5		4,5																													
50	1	Establecimiento de requisit	0,75			0,75																												
51	0	Diseño de clases y esquema	1,5			1,5																												
52	0	Creación y comienzo de la ir	3,5			3,5																												
52	0	Avanzada implementación d	5													5																	
52	0	Progreso con los métodos d	5,5														5,5																	
52	0	Progreso con función edit d	5,25															5,25																
52	0	Terminada la vista y avanza	5,25																5,25															
52	0	Terminado el modelo y avar	5																	5														
52	0	Corregidos algunos detalles	5																				5											
52	0	Avanzado el método de vist	5																					5										
52	0	Corrección de errores y ava	6																						6									
52	0	Mejorado el control de erro	4,5																							4,5								
52	0	Terminada la funcionalidad	5																								5							
52	0	Añadida la funcionalidad de	1,25																										1,25					
50	2	Análisis de librerías alternat	2,5																										2,5					
52	0	Modificadas las cajas de con	5,5																											5,5				
52	0	Añadida funcionalidad de at	4,75																												4,75			
50	3	Análisis de librerías alternat	1																													1		
52	0	Corregidos algunos casos qu	4,25																													4,25		
52	0	Terminado de implementar	4,75																															4,75
		Esfuerzos totales mes	91,75	6	4,5	5,75	0	0	0	0	0	0	0	0	0	5	5,5	5,25	5,25	5	0	0	5	5	6	4,5	5	0	0	3,75	5,5	4,75	5,25	4,75

Tabla 9.2. Esfuerzos en Julio de 2015



Tarea	Subtarea	Descripción de la actividad	Total	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
52	0	Terminada la funcionalidad d	6,75			6,75																												
52	0	Corregido el objeto JSON qu	2				2																											
50	4	Cambiado de librería Scroller	6,25					6,25																										
52	0	Scroll horizontal funcionand	4						4																									
52	0	Cambios en la edición del da	6									6																						
52	0	Corrección de errores y añad	4,75										4,75																					
52	0	JS - AJAX: Ya se hace una úni	3,75											3,75																				
52	0	Añadido responsive a .click()	5												5																			
52	0	Comenzado el módulo de ev	3,75													3,75																		
52	0	Terminado módulo específi	5														5																	
52	0	Avanzado el módulo de Estad	5,75															5,75																
52	0	Terminado el módulo de Est	5																5															
52	0	Terminada funcionalidad de	5																	5														
52	0	Avanzada la funcionalidad de	5,5																					5,5										
52	0	Avanzada la funcionalidad de	5,25																							5,25								
52	0	Terminada la visualización d	5																											5				
52	0	Ahora los widgets tienen un	5																															5
		Esfuerzos totales mes	83,75	0	0	6,75	2	6,25	4	0	0	0	6	4,75	3,75	5	3,75	0	0	5	5,75	5	5	5,5	0	0	5,25	0	0	0	5	0	0	5

Tabla 9.3. Esfuerzos en Agosto de 2015

Tarea	Subtarea	Descripción de la actividad	Total	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
52	0	Añadido módulo de Fichero	5,25	5,25																													
54	1	Creadas las ayudas para el u	0,5		0,5																												
54	2	Documentación para los des	0,5		0,5																												
55	1	Configuración de máquina v	4		4																												
55	1	Configuración de máquina v	4			4																											
55	2	Configuración de NetBeans	0,75			0,75																											
53	1	Creado esqueleto del test d	0,25			0,25																											
53	1	Implementados algunos tes	5				5																										
53	1	Test preloadjs omitido	0,25							0,25																							
53	2	Empezados tests con Seleni	5							5																							
53	2	Avanzado test de la pantalla	5								5																						
53	2	Tests para la pantalla de con	3,5									3,5																					
53	2	Terminados tests básicos de	4										4																				
53	1	Modificados los tests de PHP	0,75										0,75																				
53	0	Terminados los tests	3														3																
		Esfuerzos totales mes	41,75	5,25	5	5	5	0	0	5,25	5	3,5	4,75	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 9.4. Esfuerzos en Septiembre de 2015

	Total	07/15	08/15	09/15
Coste en horas	217,25	91,75	83,75	41,75

Tarea	Descripción	Horas
50	Análisis	10,5
51	Diseño	1,5
52	Implementación	158,25
53	Pruebas unitarias	26,75
54	Documentación/Manuales	1
55	Preparación entorno de desarrollo	19,25
		217,25

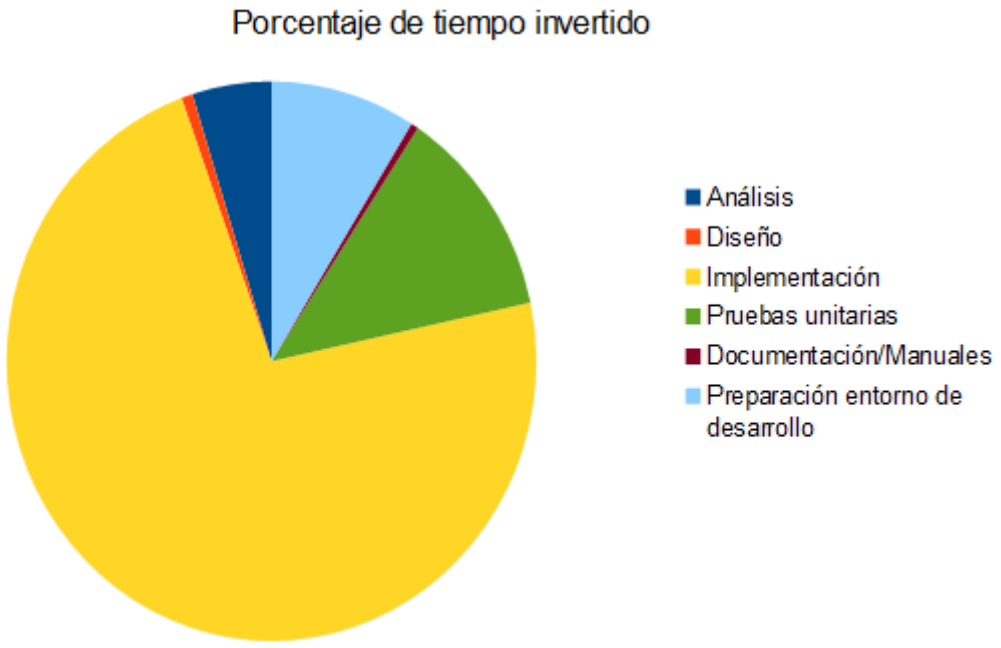


Tabla 9.5. Resumen de esfuerzos totales

Anexo VIII. Prácticas previas al TFG

Durante 6 meses, previamente a la realización del TFG, trabajé en prácticas para adaptarme al *framework* propio desarrollado por Iternova. En ese período contribuí en gran medida al análisis, captación e introducción de datos en el sistema *SmartRoads*. Mis tareas consistían en geolocalizar e introducir elementos en la base de datos del sistema, analizando imágenes tomadas en las carreteras. De esta manera el personal que hace labores de gestión y conservación de las carreteras tienen almacenados cada uno de los elementos que influyen en estas carreteras en el sistema para poder trabajar con ellos.

Así mismo, participé en el desarrollo de *scripts* para la automatización de tareas en la herramienta de integración continua *Jenkins*, usando *Apache Ant* (basado en *XML*). En estos *scripts* se incluían la actualización y distribución de una versión del sistema a través de la red local de los desarrolladores. De esta manera todos los desarrolladores disponen de la versión más actualizada del sistema. Para ello era necesario, entre otras cosas, ofuscar (codificar en binario, usando *ionCube*) el código de la nueva versión, comprimirla (para reducir el tráfico en la red), mandarla a todos los ordenadores de los desarrolladores en la red local a través de *SFTP*, comprimir la versión actual de los ordenadores (para guardarla como *backup*) y descomprimir el código ofuscado de la nueva versión. Si ha fallado alguno de estos pasos se repite hasta que no falle (por error en el envío a través de la red de los archivos comprimidos, o cualquier otro motivo). Posteriormente se descomprime la nueva versión y se decodifica (usando *ionCube* de nuevo). Por último se actualizan los enlaces simbólicos que apuntan a la última versión del sistema.

Por último, también desarrollé un nuevo módulo que usaba los *logs* de acciones de los usuarios del sistema con el fin de mostrar datos estadísticos sobre la cantidad de veces que accede un usuario a un módulo o la cantidad de veces que se ha accedido a un módulo en general.